Fork in the Bill: a pre-registered feature-flag experiment on a coding agent's model routing

Fork in the Bill team (CoffeeAndToggle)

Toggle Wars hackathon, Track 01 (the project is also entered in Tracks 02 to 04). Source: https://github.com/Prashant-thakur77/fork-in-the-bill. Live board: https://forkbill.prashantthakur.dev

ABSTRACT

A coding agent spends many of its tokens on work that needs little judgement: searching a repository, reading long files, running tests and reading the log. We asked whether moving that work from a frontier model to a small one lowers the dollars spent per solved task. The agent has two lanes. A planner on Claude Sonnet 5.5 decides and edits, and a worker lane does the grunt work. In the `control` arm the worker is also Sonnet; in `lanes_v1` it is Claude Haiku 5.5. Flagsmith assigned every task to an arm through an experiment on one multivariate flag, recorded the exposure, received the outcome events and computed the verdict our decision rule reads. The hypothesis, metrics, margin and decision rule were hashed, tagged and pushed before the experiment started. On 76 control and 69 lanes_v1 SWE-bench-Live tasks, the small worker changed mean cost per task by −56.7% and cost per solved task from $0.1753 to $0.0837. The solve rate moved from 44.7% to 40.6%, a difference of −4.2 pp with a posterior interval of −19.8 pp to +11.8 pp. All three pre-registered conditions held, so `lanes_v1` shipped. Every recorded arm was re-derived offline from Flagsmith's own evaluation engine, and every figure in this paper is filled from one JSON file and checked by script. The experiment rules out large solve-rate losses only; a drop of a few points would go undetected at this size. A registered replication on 61 control and 60 lanes_v1 tasks that no earlier run had used gave the same decision: cost per solved task changed by −51.7% and the solve rate by −2.4 pp, and the decision was applied in Flagsmith by script.

1 Introduction

Agent frameworks often let a main model hand sub-tasks to a helper. If the helper runs on the same frontier model, the cheap work is billed at the frontier price. The gap is large. In the pricing table this project uses (`harness/pricing.json`, checked on 2026-11-27), Sonnet 5.5 costs $2 per million input tokens and $10 per million output tokens, and Haiku 5.5 costs $0.10 and $0.50. In our pilot (`prereg/pilot-report.md`), the worker lane took 61% of the all-Sonnet arm's cost.

Moving the worker to the small model is the obvious saving. What it does to the number of tasks the agent solves is less obvious, and a lower solve rate can eat the saving: if each task is cheaper but fewer are solved, the cost per solved task can stay flat or rise. Published routing results come from other agents and other task sets, so they don't settle this for a given harness. The direct way to find out is a controlled experiment on real tasks where something other than the person running it picks the arm.

This paper reports experiment 1 of that study, and in sections 7 and 8 a follow-up experiment and a replication. Its contributions are practical:

- a two-lane agent harness whose arms differ in one field of a JSON policy served by a feature flag;
- a pre-registered decision rule that reads the flag platform's own verdict and a separate replication;
- an integrity trail (hash-chained ledger, environment snapshots and offline re-derivation of every assignment) that a reader can check with one command;
- the result: the small worker halved the cost per solved task with no large solve-rate loss.

2 Related practice

Randomised online experiments are the standard way product teams decide whether a change helps. A unit (a user, a session, here a task) is hashed into an arm, an exposure is logged at the point the arm takes effect, and outcomes are joined back to the exposure. Two habits from that practice matter here. One is the guardrail metric, which a change must not harm even when it wins on the primary metric. The other is the sample-ratio check: if the arms receive a different share of units than the split says, something in the pipeline is broken and the result shouldn't be trusted.

Pre-registration comes from clinical and social science. The analysis plan is fixed and published before data arrive, so that choices like the metric, the margin or when to stop can't be tuned to the result. Repeatedly checking a running experiment and stopping when it looks good inflates false positives; a fixed stopping rule avoids that. A guardrail framed as "no worse than a margin" is a non-inferiority test, and the margin has to be justified by the sample size the study can afford.

Model routing and cascades send each request, or each part of a request, to the cheapest model that can handle it. Orchestrator and worker designs, where a strong model plans and smaller models carry out narrow steps, are one form of this. Claims about these designs are usually made on benchmark suites with the routing chosen by the authors. Our setup is narrower: one harness, one model pair, and a flag platform choosing the arm per task.

3 Method

3.1 Units and task pools

A unit is one SWE-bench-Live MultiLang task: a real issue in a C, C++, C#, Go, Java, JavaScript, Rust or TypeScript repository, with tests that fail before the fix (FAIL_TO_PASS) and tests that must keep passing (PASS_TO_PASS). Tasks were drawn from one difficulty stratum: gold patches touching at most 3 files and 60 changed lines, problem statements under 12,000 characters (DECISIONS.md, D10). Pools are drawn by a seeded selection (seed 20261127) and are disjoint: pilot, experiment 1, rollout and experiment 2 never share a task. Each task is validated arm-blind before it is enqueued: its gold patch is applied and its tests run, with no model involved.

The registered pool held every task in the stratum created on or after 2026-07-01, after the models' training cutoff. Each task is one Flagsmith identity, `swelive:<instance_id>`, with the traits `repo`, `language`, `created_month`, `gold_files` and `gold_lines`.

3.2 Arms

Both arms use the same planner and the same harness. They differ in one field of the flag value, `delegate.model`.

Table 1. Arms. Each variation's value is the full routing policy as JSON (`flagsmith/policies/`).

Variation key	Planner (main lane)	Worker (delegate lane)
`control` (the flag's own value)	`claude-sonnet-5-5`	`claude-sonnet-5-5`
`lanes_v1`	`claude-sonnet-5-5`	`claude-haiku-5-5`

Prompts, tools and tool order, effort per lane, token limits and the per-task caps (40 planner turns, $0.60 and 1200 seconds of agent time) are identical. `RoutingPolicy.clean_ab_diff` checks this, and `flagsmith/setup.sh` refuses to create a variant that differs in anything else.

3.3 Assignment through Flagsmith

The runner never picks an arm. `harness/policy.py` calls `get_experiment_flag("agent_routing", identifier=..., traits=...)` once per task, which returns the variant and records the exposure, and flushes the event at once (D9). The Flagsmith experiment (id 822) splits `agent_routing` 50/50 at 100% rollout, which Flagsmith implements as a system segment override on the flag; the flag is locked while the experiment runs. A separate boolean flag, `routing_kill`, forces the control policy with no exposure and no events. The registration allows it for safety only and requires every use to be logged; the experiment 1 ledger has no kill-switch record.

3.4 Metrics

After grading, `harness/events.py` sends four events under the same identifier as the exposure, so Flagsmith can join them.

Table 2. Metrics as defined in Flagsmith (`flagsmith/metrics.json`).

Metric	Aggregation	Direction	Value	Role
`task_cost`	mean	down	US dollars, every exposed task	primary
`task_solved`	occurrence	up	1, only when every FAIL_TO_PASS and PASS_TO_PASS test passes in a fresh grading container	guardrail
`task_turns`	mean	informational	planner turns	secondary
`task_seconds`	mean	informational	seconds from assignment to end of grading	secondary

Flagsmith has no ratio metric. Cost per solved task is mean cost divided by solve rate, computed in `analysis/analyse.py`.

3.5 Pre-registration and amendment

`prereg/experiment-1.md` holds the question, hypothesis, arms, metrics, decision rule, power analysis, exclusions and stopping rule. Its SHA-256 (`44d715bd6bd7f8580e8bb1950a21ea852dfea4daaf040c84009f391bf56e19c2`) is in `prereg/HASHES.txt`. It was committed at 2026-11-26 22:57:35 UTC, tagged `prereg-exp1`, and passed CI on GitHub (run 45908009314) about five minutes before the Flagsmith experiment started at 23:02:34 UTC. The hypothesis text was pasted unchanged into the experiment's `hypothesis` field.

The registration missed one constraint. Flagsmith shows no inference for a metric until each variant has 50 identities, and the fresh pool was expected to give about 43 per arm, so the first condition of the decision rule could never have passed. Amendment 1 (`prereg/experiment-1-amendment-1.md`, tag `prereg-exp1-amend1`, committed 23:14:42 UTC) added the next 80 tasks created between 2026-03-01 and 2026-06-30, from the same stratum and the same seeded order. When it was written three tasks had finished, and the author had seen only exposure and event counts, with no result per arm and no cost. The arms, flag values, caps, metrics, margin, decision rule and stopping rule did not change. The amendment also asks for the fresh and pre-cutoff tiers to be reported separately as a secondary analysis outside the decision.

3.6 Decision rule

Ship `lanes_v1` only if all three hold; otherwise keep `control` and publish the reason.

1. Flagsmith reports a chance to beat control of at least 95% on `task_cost`.
2. Our `Beta(1,1)` posterior gives $P(\Delta\text{solve} > -m) \geq 0.90$, with $m = 20$ percentage points, where Δsolve is the lanes_v1 solve rate minus the control solve rate.

3. Flagsmith does not report `task_solved` as losing (chance to beat control below 5%).

There is no early stopping. The run ends when every task in both pools is done, excluded or invalid.

3.7 Power, and the margin chosen after the pilot

The pilot ran 9 tasks under each policy, forced by the runner outside any experiment. From it, `analysis/power.py` (2000 simulations, seed 20260202) gave the figures in the amendment: at 70 tasks per arm, a 25% cut in mean cost is detected with 80% power, a solve-rate drop of 20 points is detected with 80% power, and an arm truly as good as control passes rule 2 87% of the time. A drop of 5 points would go undetected; ruling it out would need roughly 800 tasks per arm.

We disclose that the margin m = 20 points was chosen after the pilot had been run and looked at, and before any experiment task ran (D19). It was set so that an arm as good as control passes the guardrail more often than not at the size the fresh pool allows. A 5-point margin would have failed almost every time even for an equal arm, so the experiment would have decided nothing. A reader who wanted a tighter margin should read rule 2 as a check against large losses only.

3.8 Exclusions

Infrastructure failures (image pull, container start, git trouble in the container, provider errors after retries) are retried once, then excluded and listed. Events are sent only for a task's final attempt. Agent failures (refusals, hitting a cap, malformed loops) are never excluded. A sample-ratio p-value below 0.001 would have stopped any action on the results until the cause was found.

3.9 Harness changes before registration

Three changes made during dry runs shaped the arms. All three apply identically to both and were in place before the pilot, so the pilot, the registration and the experiment use the same harness.

- D16: the shared planner prompt tells the agent to delegate surveys, long reads and test runs. In the dry run before it, the planner never delegated in four tasks.
- D17: the planner's shell output was cut to the first and last 1 KB. It still delegated nothing; it switched to tighter `sed -n` and `grep | head` commands.
- D18: the planner lost its shell. It has `view_file`, `edit_file`, `delegate` and `submit`; the worker has `bash` with 16 KB of output in the same container.

Without these the two arms would have run the same model on nearly every call and measured nothing. The result applies to this planner and worker design, not to an agent that does its own grunt work.

3.10 Analysis

Our replication uses different methods from Flagsmith's on purpose (D12). Flagsmith approximates the relative lift as normal (delta method) and reports a chance to win and a 95% interval. We use a percentile bootstrap for mean cost and cost per solved task, and independent `Beta(1,1)` posteriors for solve rate. When the two agree, the result is less likely to be an artefact of one method. The sample-ratio check is a chi-square test against the 50/50 split.

4 Integrity

Each run writes a hash-chained ledger (`runs/<exp>/ledger.jsonl`). Every record's hash is `sha256(prev_hash + canonical_json(payload))`, so editing or deleting any record breaks every hash after it. The ledger holds task completions, events sent, exclusions, environment snapshots, configuration checks and kill-switch uses. A Merkle root over the `task_done` records (raw 32-byte leaves, an odd node paired with itself) gives one value to publish and compare.

`integrity/snapshot_env.py` saves the Flagsmith environment document at the start, every 15 minutes and at the end of a run (D21), and writes each snapshot's hash to the ledger. A change to the flag or the split during the run would show as a second distinct hash.

Flagsmith assigns a variant by hashing the identity, so the arm each task should have run follows from the environment document. `integrity/verify_assignment.py` evaluates every identity in the run against the saved start-of-run document with Flagsmith's `flag-engine`, the code the SDK's local evaluation uses, and fails if any recorded arm differs.

Every figure in this paper is a marker filled from `analysis/out/numbers.json` by `analysis/render_docs.py`. `analysis/check_consistency.py` fails CI if any marker is stale or if a bare number appears inside a results block.

For `exp1`, the Merkle root over 146 `task_done` records is `7ed364648c7513ef22280eaed97a2ab77e92d91b8f88f5bbfab08d747a989929`. The run took 56 environment snapshots with 1 distinct hash. Flagsmith's engine (version 11.1.2), run offline against the start-of-run document (SHA-256 prefix 9f01eed3330f), reproduced 146 of 146 recorded arms, with 0 mismatches.

The ledger holds one more `task_done` record than there are analysed tasks. One identity, `swelive:gui-cs__Terminal.Gui-4860`, ran twice: its second attempt started after the first had finished. Both attempts ran `lanes_v1` and both were unsolved. The analysis keeps the last attempt of each identity.

A reader who wants to check one record without the whole ledger can use an inclusion proof, as in Certificate Transparency: the record's hash and the sibling hashes on its way up the tree, which rebuild the root. `integrity/prove.py` prints and checks one, and the proofs of every record are published with the analysis. 146 of 146 records verify against the published root, each with at most 8

hashes. The tree keeps the rule the root was published with (no leaf or node prefix, an odd node paired with itself), which differs from RFC 6962's, so proofs are checked against the published tree size as well as the root.

5 Results

All figures in this section come from `analysis/out/numbers.json` (Flagsmith results pulled at the time shown below, saved raw under `analysis/flagsmith/exp1/`).

Main table

Table 3. Experiment `exp1`, `lanes_v1` against `control`. Intervals are percentile bootstrap intervals for costs and posterior intervals for solve rates.

	`control`	`lanes_v1`
Tasks exposed	76	69
Excluded	0	0
Analysed	76	69
Mean cost per task	$0.0784	$0.0340
Mean cost, interval	$0.0632 to $0.0962	$0.0282 to $0.0406
Median cost per task	$0.0491	$0.0273
Solves	34	28
Solve rate	44.7%	40.6%
Solve rate, interval	34.1% to 55.9%	29.8% to 52.4%
Cost per solved task	$0.1753	$0.0837
Cost per solved task, interval	$0.1233 to $0.2607	$0.0614 to $0.1191
Worker share of cost	62.5%	11.5%
Worker share of tokens	55.3%	64.8%
Mean planner turns	4.921	5.145
Mean seconds, assignment to graded	623	442

Bootstrap intervals use 10,000 resamples and posterior quantities use 200,000 draws.

Cost

The small worker changed mean cost per task by −$0.0444 (−56.7%), with a bootstrap interval of −$0.0632 to −$0.0281 (−67.1% to −42.9%). Cost per solved task is mean cost divided by solve rate: $0.0784 ÷ 44.7% = $0.1753 for `control` and $0.0340 ÷ 40.6% = $0.0837 for `lanes_v1`, a change of −52.2% (interval −71.1% to −22.3%).

The worker's share of tokens went up under `lanes_v1` while its share of cost fell from 62.5% to 11.5%. The saving comes from the price of the worker's tokens, not from the agent doing less.

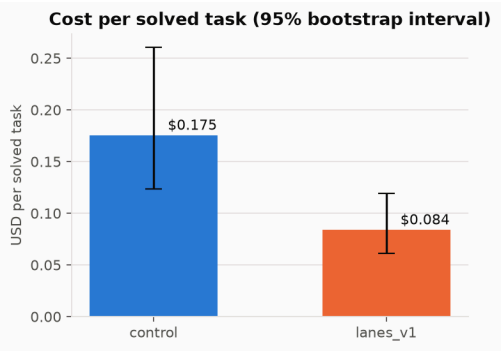


Figure 1. Cost per solved task by arm, with bootstrap intervals.

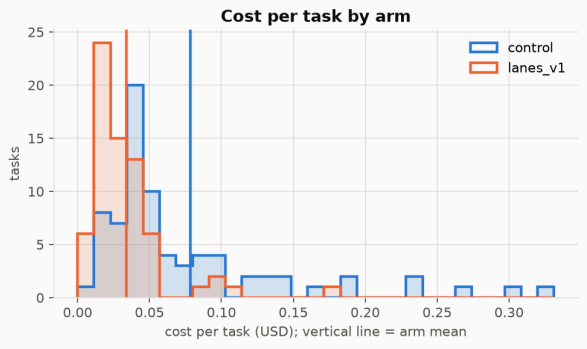


Figure 2. Distribution of cost per task by arm.

Solve rate (guardrail)

`lanes_v1` solved 28 of 69 tasks and `control` solved 34 of 76. The difference is −4.2 pp, with a posterior interval of −19.8 pp to +11.8 pp. The posterior probability that `lanes_v1` solves fewer tasks than `control` is 69.2%, so a small loss is more likely than not. The probability that the drop is smaller than the margin m = 20.0% is 97.7%, above the bar of 90.0%.

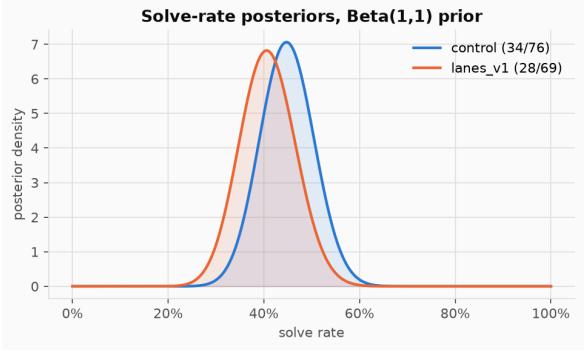


Figure 3. Solve-rate posteriors for each arm under a flat prior.

Flagsmith's verdict and our replication

Table 4. Flagsmith's readout next to our replication of the same quantities.

Quantity	Flagsmith	Ours
Relative change in mean cost	−56.6%	−56.7%
Interval on that change	−68.8% to −44.3% (normal)	−67.1% to −42.9% (bootstrap)
Chance to beat control, `task_cost`	100.0%	
`control` mean cost	$0.0784	$0.0784
`lanes_v1` mean cost	$0.0341	$0.0340
`control` solve rate	44.7%	44.7%
`lanes_v1` solve rate	40.6%	40.6%
`task_solved` status	inconclusive	

The two mean costs differ by $0.0000 for `control` and $0.0001 for `lanes_v1`; the solve rates match exactly. Flagsmith's results were pulled at 20261127T120242Z. The sample-ratio check gives chi-square 0.338 and p = 0.561 (flagged: False).

Decision

Table 5. The pre-registered rule, condition by condition.

Condition	Bar	Met
Flagsmith chance to beat control on `task_cost`	at least 95.0%	True
Posterior P(Δsolve > −m)	at least 90.0%	True
Flagsmith does not report `task_solved` as losing		True

Shipped: lanes_v1.

Secondary readings

Turns and wall time are informational. `lanes_v1` used slightly more planner turns on average (5.145 against 4.921) and less wall time (442 against 623 seconds). Wall time includes image pulls, builds and grading, which dominate it.

The registered CUPED adjustment, which uses a pilot-trained cost covariate, ran as a secondary analysis and changed little: variance reduction +3.4%, adjusted difference −$0.0448. It is labelled secondary and does not enter the decision.

By difficulty tertile, `lanes_v1` cost less in every tertile. Its solve rate was lower in the low and mid tertiles (58.3% against 68.0%, and 40.9% against 46.4%) and higher in the high tertile (21.7% against 17.4%). Each cell holds a few dozen tasks or fewer, so these splits describe the sample and test nothing. The full tables are in the appendix.

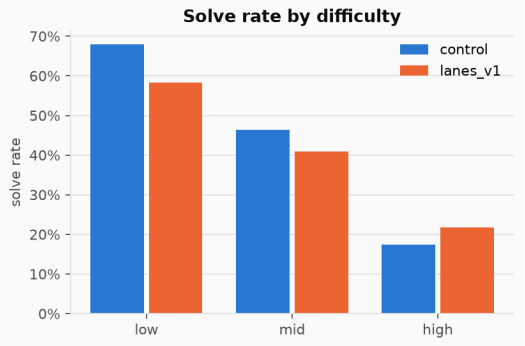


Figure 4. Solve rate by difficulty tertile.

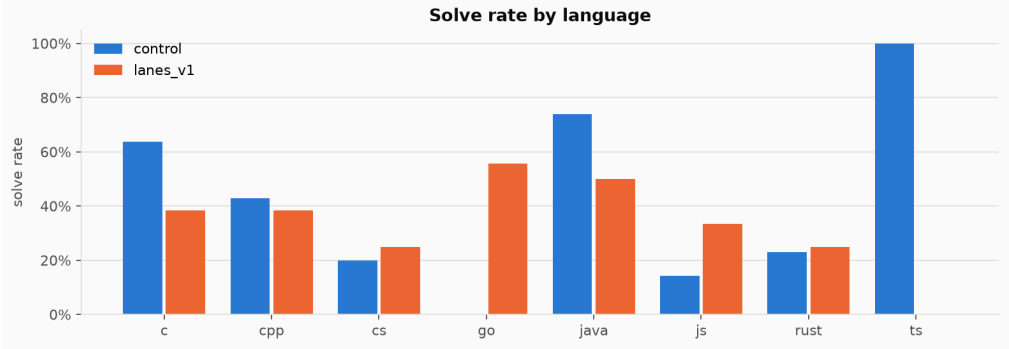


Figure 5. Solve rate by language.

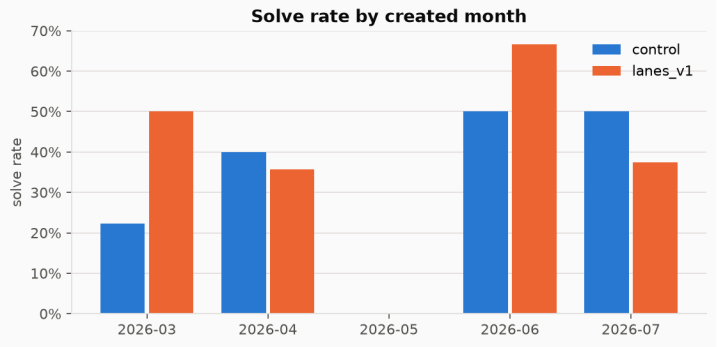


Figure 6. Solve rate by the month the task was created.

The amendment asked for the fresh tier (created in or after July) and the pre-cutoff tier to be reported separately, as a secondary analysis that does not enter the decision. On the fresh tier alone (54 control and 40 lanes_v1 tasks), `lanes_v1` solved 37.5% against 50.0% for `control`: a difference of −12.5 pp, posterior interval −31.1 pp to +7.9 pp. The posterior probability that this drop is within m is 78.9%, below the 90.0% bar that the guardrail clears on the full pool. On the pre-cutoff tier (22 and 29 tasks) the difference runs the other way, +13.0 pp (interval −14.0 pp to +36.5 pp). Cost per solved task changed by −40.9% on fresh tasks and −70.3% on pre-cutoff tasks. The registered decision uses both tiers together, and neither tier alone is large enough for a verdict. But a reader who cares only about tasks the models cannot have seen should take the fresh-tier figures: the saving holds there, and a real solve-rate loss of ten points or more is not ruled out.

Robustness checks added after the decision

Four analyses were added after experiment 1 had ended and been decided (DECISIONS.md, D29). They are secondary, none was registered, and none changes the decision above; they run for every experiment whose logs are committed.

The live board showed running totals per arm once it went public, by which time 94 tasks had finished (D24), so we asked what a reader watching it could validly have concluded. A mixture sequential probability ratio test with a normal mixing distribution gives a p-value that stays valid however often it is read [14]. Its mixing scale was fixed from the pilot alone, before the experiment's

trajectory was computed: $0.0420 per task for cost and 11.1% for solve rate. Replayed in the order the 145 analysed tasks finished, the always-valid p-value for mean cost first fell below 0.050 after task 72 and ended at <0.001, with an always-valid interval of −$0.0709 to −$0.0172 per task. For solve rate it ended at 0.630 (interval −31.2 pp to +18.4 pp). Whether it crossed the line at any point: False; with `lanes_v1` solving less: False. So a watcher could have seen the cost saving without the bias that peeking brings, and the running solve rates gave no valid signal either way.

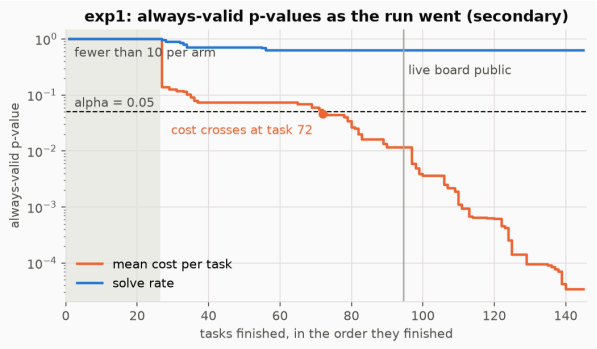


Figure 7. Always-valid p-values for mean cost and solve rate, in the order tasks finished.

Following the case for reporting an agent's cost next to its accuracy [9], every arm the project ran is on one cost-accuracy plane: the pilot's two arms, both arms of experiment 1 and the rollout batch. 3 of 9 arms are on the Pareto frontier (exp3 lanes_v1, rollout lanes_v1, exp2 control (lanes_v1 policy)). The batches used different task pools, so the plane describes them; only the two arms of experiment 1 are a controlled comparison.

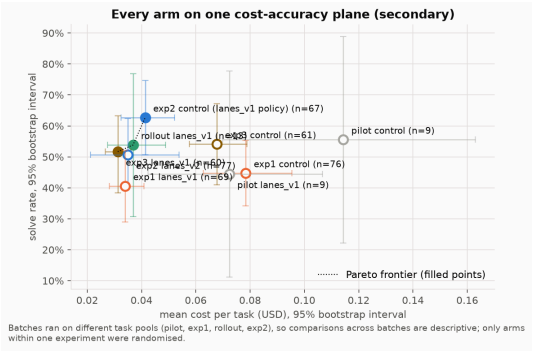


Figure 8. Mean cost per task against solve rate for every arm, with bootstrap intervals. Filled points are on the frontier.

Last, the garden of forking paths [17]: we recomputed the change in cost per solved task and the guardrail posterior for every combination of task set (all, fresh tier, pre-cutoff tier), the attempt kept for the repeated identity, the cost statistic (mean, median, trimmed mean), the solve-rate prior (flat or Jeffreys) and cost outliers (as recorded or winsorised). Over 72 specifications, cost per solved task is lower under `lanes_v1` in 72, ranging from −70.3% to −31.2%. The guardrail posterior clears its bar in 48 (66.7%) and ranges from 77.9% to 99.2%. Every specification that fails it uses the fresh tier alone (24 of 24), which is the by-tier result above seen from another side: the cost saving does not depend on the analysis choices, and the guardrail depends on whether the pre-cutoff tasks are counted.

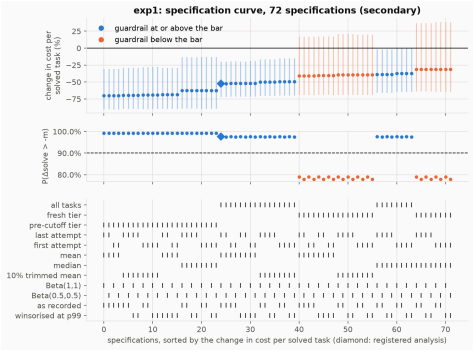


Figure 9. Specification curve. Top: change in cost per solved task with bootstrap intervals, sorted. Middle: the guardrail posterior against its bar. Bottom: the choices behind each specification. The diamond is the registered analysis.

Total model spend across the study, including dry runs and the pilot, was $23.34, of which `exp1` used $8.52.

6 Rollout on the same flag

After the decision, Flagsmith experiment 822 was ended with `lanes_v1` as the rollout decision, so `lanes_v1` became the value of `agent_routing` itself at 100%. The experiment was ended at 2026-11-27 12:05 UTC. No code changed. A fresh batch from `tasks/pools/rollout.txt`, disjoint from every other pool, then ran on the same flag with `make rollout`: the winning policy served as the flag's own value, outside any experiment, on tasks the experiment never saw.

The batch ran 13 valid tasks at a mean cost of $0.0369 per task and solved 53.8% of them. Every task got `lanes_v1` from the flag, which the offline re-derivation confirms. This batch is small and has no comparison arm, so it shows the rolled-out policy working as the flag's value at a cost in line with the experiment, not a second estimate of the effect.

7 Experiment 2: a draft-review worker role

Experiment 2 starts from the rolled-out state. Its control is the experiment 1 winner, now the flag's own value, and its treatment `lanes_v2` differs from it only in `prompt_variant` (`delegate_v1` becomes `draft_review_v1`, a draft-review role for the worker, which switches the planner's and the worker's prompts and gives the worker file-editing tools); the models and caps are unchanged (`flagsmith/policies/lanes_v2.json`, `flagsmith/experiment-2.json`). It was registered, hashed and tagged (`prereg-exp2`) before its first task, its pool was sized to clear Flagsmith's 50-identity floor per variant (D22, D26), and it ran as Flagsmith experiment 1423.

The run analysed 67 control and 77 `lanes_v2` tasks. Mean cost per task changed by −15.4% (bootstrap interval −51.5% to +39.2%), and the solve rate went from 62.7% to 50.6%, a difference of −12.0 pp with a posterior interval of −27.3 pp to +4.3 pp. Cost per solved task therefore rose by +4.8%: the drafts were cheaper to produce, and fewer of them ended in a fix. Flagsmith put the chance that `lanes_v2` is cheaper per task at 72.6%, and the posterior probability that the solve-rate drop is within the margin was 84.8%. Neither reached its registered bar, so the rule kept control. The sample-ratio check did not flag (p = 0.455).

The decision was applied by `analysis/decide.py`, which reads the rule's outcome from the analysis output, checks the registration's hash, ends the Flagsmith experiment with that rollout decision and appends the call's result to the run ledger. The flag kept serving the `exp1` winner to every identity.

8 Experiment 3: a replication of experiment 1

One experiment on one pool can be a fluke of that pool. Experiment 3 asks experiment 1's question again with the same arms, metrics, margin and decision rule, on the next 183 tasks in the same seeded order, none of which an earlier run had used (`prereg/experiment-3.md`, tag `prereg-exp3`). It was decided on 28 November after experiments 1 and 2 had ended; nothing in its design was chosen from experiment 1's data except the expected effect size used for its power statement. For it the flag's own value was set back to the all-Sonnet `control` policy, so Flagsmith split identities between `control` and `lanes_v1` as in experiment 1. It ran as Flagsmith experiment 1774. The registration defines success as the rule shipping `lanes_v1` again.

The run analysed 61 control and 60 `lanes_v1` tasks. Mean cost per task changed by −53.9% (bootstrap interval −62.9% to −42.8%), and cost per solved task went from $0.1253 to $0.0605 (−51.7%, against −52.2% in experiment one). The solve rate went from 54.1% to 51.7%, a difference of −2.4 pp with a posterior interval of −19.7 pp to +15.1 pp, and the posterior probability that the drop is within the margin was 97.7%. Flagsmith put the chance that `lanes_v1` is cheaper per task at 100.0% and did not report `task_solved` as losing. The sample-ratio test gave p = 0.787. All three conditions held, so the rule shipped lanes_v1 and the replication succeeded by the registration's own definition.

Pooled across both experiments, a secondary reading the registration announced, the 266 tasks give a change in cost per solved task of −52.5% (bootstrap interval −65.9% to −33.2%) and a solve-rate difference of −3.2 pp (posterior interval −14.9 pp to +8.8 pp). The pools differ in task age, so the pooled figures summarise the evidence and do not replace either experiment's decision.

`analysis/decide.py` applied the decision: it ended the Flagsmith experiment with `lanes_v1` as the rollout decision and appended the result to the run's ledger. Fresh identities then received `lanes_v1`, and the policy was made the flag's own value with the variant weights at zero, as after the first experiment.

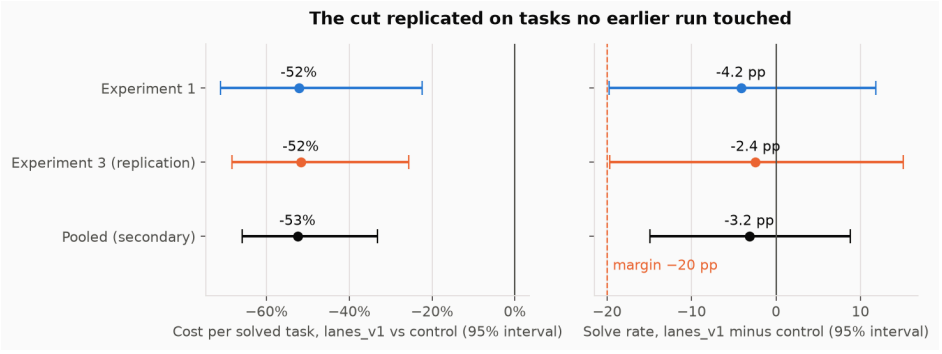


Figure 10. Experiment 1, experiment 3 and the pooled estimate on both registered metrics

9 Threats to validity and limits

The margin was set after the pilot. m = 20 points was chosen with the pilot's numbers in view (section 3.7), before any experiment task ran. The guardrail rules out large losses only, and the point estimate leans toward a small loss.

The amendment changed the population mid-registration. It added pre-cutoff tasks that the models may have seen in training. Assignment is random, so contamination affects both arms alike in expectation, but it can raise absolute solve rates. The amendment was written after three tasks had finished, on counts only.

One harness, one model pair, one stratum. The planner has no shell by design (D18). The tasks are small changes (at most 3 files and 60 lines) in eight non-Python languages. Other model pairs, a stronger planner, larger changes or an agent that does its own grunt work could behave differently.

Prices. All calls went to one provider at list prices on one date. Cached-token prices and price changes move the dollar figures, though not the token shares.

Platform. Flagsmith's experimentation feature is in beta and its results response is not documented. `analysis/pull_results.py` parses the shape seen on Flagsmith Cloud and saves every raw response for re-checking.

Repeated attempt. One identity ran twice (section 4); the analysis keeps its last attempt. Both attempts were in the same arm and both failed, so the choice does not change any rate.

10 Reproducibility

The run logs (`runs/<exp>/*.jsonl`), ledgers, environment snapshots and raw Flagsmith results are committed. From a clean clone, after `uv sync`:

```
make analyse                # numbers.json, charts and results.md from the logs, then the consistency gate
uv run python -m integrity.verify_ledger runs/exp1         # hash chain and Merkle root
uv run python -m integrity.verify_assignment --run runs/exp1  # every arm re-derived with flag-engine
sha256sum -c prereg/HASHES.txt                              # registration files match their hashes
uv run python -m analysis.check_consistency                 # every figure in the docs matches numbers.json
make verify                 # ledger checks for every committed run
make docs                   # fill this paper and the deck from numbers.json, then build the PDFs
```

No API key is needed for any of these. Running the experiment again needs Flagsmith, a model API key and Docker; the README lists the setup and every Flagsmith CLI call.

11 Conclusion

With the planner fixed on Sonnet 5.5, moving the worker lane to Haiku 5.5 cut the cost per solved task from $0.1753 to $0.0837 on real SWE-bench-Live tasks. The solve rate changed by −4.2 pp at the point estimate, inside a margin that rules out large losses but not small ones. Flagsmith chose every arm, its verdict and our replication agree on cost to within a hundredth of a cent, and every assignment re-derives offline. The winner now runs as the flag's value. A fresh batch ran on it with no code change; a draft-review prompt for the worker (experiment 2) lost and was not shipped; and the same question asked again on unused tasks (experiment 3) gave the same answer, about half the cost per solved task with the solve-rate check passed.

Appendix A: results by trait

These tables describe the sample. Cells are small and none of them was a pre-registered test.

Table 6. By difficulty tertile.

Level	control n	control solve rate	control mean cost	lanes_v1 n	lanes_v1 solve rate	lanes_v1 mean cost
low	25	68.0%	$0.0700	24	58.3%	$0.0288
mid	28	46.4%	$0.0590	22	40.9%	$0.0387
high	23	17.4%	$0.1112	23	21.7%	$0.0349

Table 7. By language.

Level	control n	control solve rate	control mean cost	lanes_v1 n	lanes_v1 solve rate	lanes_v1 mean cost
c	11	63.6%	$0.0982	13	38.5%	$0.0462
cpp	7	42.9%	$0.0834	13	38.5%	$0.0287
cs	10	20.0%	$0.0896	4	25.0%	$0.0240
go	4	0.0%	$0.1135	9	55.6%	$0.0372
java	23	73.9%	$0.0417	18	50.0%	$0.0262
js	7	14.3%	$0.0677	3	33.3%	$0.0280
rust	13	23.1%	$0.1122	8	25.0%	$0.0444
ts	1	100.0%	$0.0542	1	0.0%	$0.0303

Table 8. By the month the task was created. Months before July are the pre-cutoff tier added by the amendment.

Level	control n	control solve rate	control mean cost	lanes_v1 n	lanes_v1 solve rate	lanes_v1 mean cost
2026-03	9	22.2%	$0.0810	12	50.0%	$0.0273
2026-04	10	40.0%	$0.0774	14	35.7%	$0.0342
2026-05	1	0.0%	$0.0775	0	n/a	n/a
2026-06	2	50.0%	$0.0776	3	66.7%	$0.0512
2026-07	54	50.0%	$0.0782	40	37.5%	$0.0346

References

Sources for the methods used here. `docs/references.md` in the repository says which part of the study each one relates to.

1. Chen, L., Zaharia, M., & Zou, J. (2023). FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. arXiv:2305.05176; accepted by Transactions on Machine Learning Research. https://arxiv.org/abs/2305.05176
2. Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., & Stoica, I. (2024). RouteLLM: Learning to Route LLMs with Preference Data. arXiv:2406.18665; ICLR 2025 (poster). https://arxiv.org/abs/2406.18665
3. Aggarwal, P., Madaan, A., Anand, A., et al. (2023). AutoMix: Automatically Mixing Language Models. arXiv:2310.12963; NeurIPS 2024. https://arxiv.org/abs/2310.12963
4. Ding, D., Mallick, A., Wang, C., Sim, R., Mukherjee, S., Ruhle, V., Lakshmanan, L. V. S., & Awadallah, A. H. (2024). Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing. arXiv:2404.14618; ICLR 2024. https://arxiv.org/abs/2404.14618
5. Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2023). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770; ICLR 2024. https://arxiv.org/abs/2310.06770
6. Zhang, L., He, S., Zhang, C., et al. (2025). SWE-bench Goes Live! arXiv:2505.23419; NeurIPS 2025 Datasets and Benchmarks Track. https://arxiv.org/abs/2505.23419
7. Li, K., Li, R., Zhang, L., et al. (2026). RepoLaunch: Automating Build and Management of Code Repositories across Languages and Platforms. arXiv:2603.05026 (arXiv comment: NeurIPS 2026 main track, poster). https://arxiv.org/abs/2603.05026
8. Liang, S., Garg, S., & Moghaddam, R. Z. (2025). The SWE-Bench Illusion: When State-of-the-Art LLMs Remember Instead of Reason. arXiv:2506.12286. https://arxiv.org/abs/2506.12286
9. Kapoor, S., Stroebl, B., Siegel, Z. S., Nadgir, N., & Narayanan, A. (2024). AI Agents That Matter. arXiv:2407.01502; accepted by Transactions on Machine Learning Research. https://arxiv.org/abs/2407.01502
10. Xu, B., Peng, Z., Lei, B., Mukherjee, S., Liu, Y., & Xu, D. (2023). ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models. arXiv:2305.18323. https://arxiv.org/abs/2305.18323
11. Anthropic (2024). Building effective agents. Anthropic Engineering, published 19 December 2024. https://www.anthropic.com/research/building-effective-agents
12. Kohavi, R., Tang, D., & Xu, Y. (2020). Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing. Cambridge University Press. https://doi.org/10.1017/9781108653985

13. Fabijan, A., Gupchup, J., Gupta, S., Omhover, J., Qin, W., Vermeer, L., & Dmitriev, P. (2019). Diagnosing Sample Ratio Mismatch in Online Controlled Experiments: A Taxonomy and Rules of Thumb for Practitioners. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19), 2156-2164. https://doi.org/10.1145/3292500.3330722
14. Johari, R., Koomen, P., Pekelis, L., & Walsh, D. (2017). Peeking at A/B Tests: Why it matters, and what to do about it. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17), 1517-1525. https://doi.org/10.1145/3097983.3097992
15. Deng, A., Xu, Y., Kohavi, R., & Walker, T. (2013). Improving the Sensitivity of Online Controlled Experiments by Utilizing Pre-Experiment Data. Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM 2013), 123-132. https://doi.org/10.1145/2433396.2433413
16. Nosek, B. A., Ebersole, C. R., DeHaven, A. C., & Mellor, D. T. (2018). The preregistration revolution. Proceedings of the National Academy of Sciences, 115(11), 2600-2606. https://doi.org/10.1073/pnas.1708274114
17. Gelman, A., & Loken, E. (2013). The garden of forking paths: Why multiple comparisons can be a problem, even when there is no "fishing expedition" or "p-hacking" and the research hypothesis was posited ahead of time. Unpublished manuscript, dated 14 November 2013. https://sites.stat.columbia.edu/gelman/research/unpublished/p_hacking.pdf
18. U.S. Food and Drug Administration (2016). Non-Inferiority Clinical Trials to Establish Effectiveness: Guidance for Industry. Center for Drug Evaluation and Research and Center for Biologics Evaluation and Research, November 2016. https://web.archive.org/web/20261117192132/https://www.fda.gov/regulatory-information/search-fda-guidance-documents/non-inferiority-clinical-trials
19. Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). Bayesian Data Analysis (3rd ed.). Chapman & Hall/CRC. https://doi.org/10.1201/b16018
20. Deng, A., Lu, J., & Chen, S. (2016). Continuous Monitoring of A/B Tests without Pain: Optional Stopping in Bayesian Testing. arXiv:1602.05549. https://arxiv.org/abs/1602.05549
21. Efron, B., & Tibshirani, R. J. (1994). An Introduction to the Bootstrap. Chapman & Hall. https://www.routledge.com/An-Introduction-to-the-Bootstrap/Efron-Tibshirani/p/book/9780412042317
22. Merkle, R. C. (1988). A Digital Signature Based on a Conventional Encryption Function. Advances in Cryptology - CRYPTO '87, Lecture Notes in Computer Science, 369-378. Springer. https://doi.org/10.1007/3-540-48184-2_32
23. Haber, S., & Stornetta, W. S. (1991). How to time-stamp a digital document. Journal of Cryptology, 3(2), 99-111. https://doi.org/10.1007/BF00196791
24. Crosby, S. A., & Wallach, D. S. (2009). Efficient Data Structures for Tamper-Evident Logging. 18th USENIX Security Symposium (USENIX Security 09). https://www.usenix.org/conference/usenixsecurity09/technical-sessions/presentation/efficient-data-structures-tamper-evident
25. Laurie, B., Langley, A., & Kasper, E. (2013). Certificate Transparency. RFC 6962 (Experimental), IETF; obsoleted by RFC 9162. https://www.rfc-editor.org/info/rfc6962
26. Tang, D., Agarwal, A., O'Brien, D., & Meyer, M. (2010). Overlapping Experiment Infrastructure: More, Better, Faster Experimentation. Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '10), 17-26. https://doi.org/10.1145/1835804.1835810
27. Flagsmith (n.d.). Experimentation (Beta). Flagsmith Docs, accessed 27 November 2026. https://docs.flagsmith.com/experimentation
28. Flagsmith (n.d.). Rollout by Percentage. Flagsmith Docs, accessed 27 November 2026. https://docs.flagsmith.com/managing-flags/rollout/rollout-by-percentage